Sparse Identification of Nonlinear Dynamics (SINDy)

Krishna Kumar

University of Texas at Austin

November 5, 2025

Overview

The Discovery Problem

Traditional Approach:

- Scientist observes system behavior
- Proposes governing equations based on physical intuition
- Validates through experiments

Modern Challenge:

- Complex systems: climate, turbulence, biological networks
- High-dimensional data available
- Governing equations unknown or intractable

The Question:

Can we discover governing equations directly from data?

What is SINDy?

Sparse Identification of Nonlinear Dynamics

Given measurements $\mathbf{u}(t) \in \mathbb{R}^n$ from a physical system, discover:

$$\frac{d\mathbf{u}}{dt} = \mathbf{f}(\mathbf{u}) \tag{1}$$

Key Insights:

1. Most physical systems have **sparse** dynamics (few active terms)
2. $\mathbf{f}$ can be represented as a linear combination of basis functions
3. Use sparse regression to identify the active terms

Why Sparsity?

- Occam's Razor: simplest explanation is often correct
- Physical laws typically involve few terms
- Example: $F = ma$, not $F = ma + 0 \cdot v^2 + 0 \cdot a^3 + \dots$

Example: Lorenz System

The Lorenz equations (unknown to us):

$$\dot{x} = \sigma(y - x)$$
$$\dot{y} = x(\rho - z) - y$$
$$\dot{z} = xy - \beta z$$

What we have: Time series data $\{(x_i, y_i, z_i, t_i)\}_{i=1}^{m}$

What we want: The equations above!

Key observation: Each equation is sparse in polynomial space

- Only 2-3 terms out of many possible polynomials
- Example: $\dot{x}$ only involves x, y (not $x^2, xy^2, \ldots$)

What is SINDy?

SINDy = **Sparse Identification of Nonlinear Dynamics**

The big idea: Discover governing equations just by looking at data.

Example scenario:

- Watch a pendulum swing
- Record position and velocity at different times
- Don't know any physics ($F = ma$, gravity, etc.)
- SINDy analyzes the data and tells you the exact differential equation

Output:

$$\frac{d^2\theta}{dt^2} = -\frac{g}{L}\sin(\theta)$$

The Core Assumption: Physics is Sparse

Most physical systems are governed by equations with **only a few important terms**.

Example: Simple pendulum

$$\frac{d^2\theta}{dt^2} = -\frac{g}{L}\sin(\theta)$$

We could try thousands of mathematical terms:

- Polynomials: $\theta^2, \theta^3, \theta^4, \ldots$
- Trig functions: $\cos(\theta), \tan(\theta), \sin^2(\theta), \ldots$
- Exponentials: $e^{\theta}, e^{-\theta}, \ldots$

But the true equation uses only **one**: $\sin(\theta)$

Sparsity: The governing equation is simple among a sea of possibilities.

Step-by-Step Guide: Step 1

Collect Data (and Get Derivatives)

What you need:

1. State of your system over time
   - For pendulum: angle θ and angular velocity ω
   - Measured at many time points t
2. Time derivatives $\dot{X}$ (how fast each variable changes)
   - Sometimes measured directly (e.g., accelerometer)
   - Often calculated numerically from data (slope between nearby points)

Result: Two data matrices

- X: State at all times (e.g., $[\theta, \omega]$)
- $\dot{X}$: Rate of change at all times (e.g., $[\dot{\theta}, \dot{\omega}]$)

Step-by-Step Guide: Step 2

Build a Library of Candidate Functions

SINDy doesn't know if the answer involves x^2, $\sin(x)$, or xy.

Solution: Propose a huge library Θ of all candidate functions.

Example: If state variables are x, y, library might include:

- Constants: 1
- Linear: x, y
- Quadratic: x^2, y^2, xy
- Cubic: x^3, x^2y, xy^2, y^3
- Trig: $\sin(x), \cos(x), \sin(y), \cos(y)$
- Other: $1/x, e^x, \log(y)$

Result: Matrix $\Theta(X)$ where each column is one function evaluated at every time step.

Step-by-Step Guide: Step 3

Set Up "Find the Coefficients" Problem

SINDy finds a combination of library functions that equals your derivative.

Linear algebra problem:

$$\dot{X} \approx \Theta(X) \cdot \Xi$$

- $\dot{X}$ = Time derivatives (what you want to explain)
- $\Theta(X)$ = Library of candidate functions (building blocks)
- Ξ (Xi) = Unknown coefficients (what you need to find)

For system with x, y:

$$\dot{x} = c_1(1) + c_2(x) + c_3(y) + c_4(x^2) + c_5(xy) + \dots$$
$$\dot{y} = d_1(1) + d_2(x) + d_3(y) + d_4(x^2) + d_5(xy) + \dots$$

Goal: Find all coefficients (c_i, d_i).

Step-by-Step Guide: Step 4

Find the Sparse Solution (The Magic)

Standard least-squares regression would find a solution, but:

- Uses all library functions a little bit
- Tiny non-zero values for every coefficient
- Accurate but impossible to interpret

SINDy's trick: Sparse regression
Instead of most accurate, find **simplest solution that is still highly accurate**.

Force as many coefficients Ξ as possible to be **exactly zero**.

Algorithm: Sequentially Thresholded Least Squares (STLSQ)

1. Solve for all coefficients
2. Find coefficients below threshold
3. Set small coefficients to zero permanently
4. Re-solve with only remaining "active" coefficients

The Final Result

After sparse regression, Ξ matrix is mostly zeros.

The few non-zero entries tell you **exactly which terms** build the governing equation.

Example: Chaotic Lorenz system
Variables: x, y, z

SINDy discovers from data alone:

$$\dot{x} = -10x + 10y$$
$$\dot{y} = 28x - y - xz$$
$$\dot{z} = -\frac{8}{3}z + xy$$

Found by identifying non-zero coefficients for terms: x, y, xz, z, xy

Key insight: Turns messy data into simple, interpretable differential equations

The SINDy Equation

Core approximation problem:

$$\dot{\mathbf{U}} \approx \mathbf{\Theta}(\mathbf{U})\mathbf{\Xi} \tag{2}$$

Data Matrix

$$\mathbf{U} = \begin{bmatrix} u_1(t_1) & \cdots & u_n(t_1) \\ u_1(t_2) & \cdots & u_n(t_2) \\ \vdots & & \vdots \\ u_1(t_m) & \cdots & u_n(t_m) \end{bmatrix}$$

m samples

n states

Library Matrix

$$\mathbf{\Theta}(\mathbf{U}) = \begin{bmatrix} | & & | \\ \theta_1(\mathbf{U}) & \cdots & \theta_\ell(\mathbf{U}) \\ | & & | \end{bmatrix}$$

ℓ candidate functions

Coefficients

$$\mathbf{\Xi} = \begin{bmatrix} | & & | \\ \xi_1 & \cdots & \xi_n \\ | & & | \end{bmatrix}$$

Sparse!

Goal: Find sparse $\mathbf{\Xi}$ such that $\mathbf{\Theta}(\mathbf{U})\mathbf{\Xi} \approx \dot{\mathbf{U}}$

Data Matrix $\mathbf{U}$

Collect measurements at m time points:

$$\mathbf{U} = \begin{bmatrix} u_1(t_1) & u_2(t_1) & \cdots & u_n(t_1) \\ u_1(t_2) & u_2(t_2) & \cdots & u_n(t_2) \\ \vdots & \vdots & & \vdots \\ u_1(t_m) & u_2(t_m) & \cdots & u_n(t_m) \end{bmatrix}_{m\times n}$$

Derivatives: Compute numerically or measure directly

$$\dot{\mathbf{U}} = \begin{bmatrix} \dot{u}_1(t_1) & \dot{u}_2(t_1) & \cdots & \dot{u}_n(t_1) \\ \dot{u}_1(t_2) & \dot{u}_2(t_2) & \cdots & \dot{u}_n(t_2) \\ \vdots & \vdots & & \vdots \\ \dot{u}_1(t_m) & \dot{u}_2(t_m) & \cdots & \dot{u}_n(t_m) \end{bmatrix}_{m\times n}$$

Methods for computing $\dot{\mathbf{U}}$:

- Finite differences: simple but noisy
- Total variation regularization: noise-robust

Library Matrix $\boldsymbol{\Theta}(\mathbf{U})$

Build library of candidate functions applied to data:

$$\boldsymbol{\Theta}(\mathbf{U}) = \begin{bmatrix} 1 & u_1 & u_2 & \cdots & u_n & u_1^2 & u_1 u_2 & \cdots \end{bmatrix}$$

Polynomial Library (degree d):

- Constant: 1
- Linear: $u_1, u_2, \ldots, u_n$
- Quadratic: $u_1^2, u_1 u_2, u_1 u_3, \ldots, u_n^2$
- Cubic: $u_1^3, u_1^2 u_2, \ldots, u_n^3$
- $\vdots$

Other Options:

- Trigonometric: $\sin(u_i), \cos(u_i)$
- Exponential: e^{u_i}
- Rational: $\frac{1}{u_i}$
- Custom: domain-specific functions

Example: 2D Polynomial Library

For $\mathbf{u} = [x, y]^T$ with degree 2:

$$\mathbf{\Theta}(\mathbf{u}) = \begin{bmatrix} 1 & x & y & x^2 & xy & y^2 \end{bmatrix}$$

For Lorenz system (x, y, z) with degree 2:

$$\mathbf{\Theta}(\mathbf{u}) = \begin{bmatrix} 1 & x & y & z & x^2 & xy & xz & y^2 & yz & z^2 \end{bmatrix}$$

Each row of $\mathbf{\Theta}(\mathbf{U})$ evaluates these functions at one time point.

Size: $m \times \ell$ where ℓ = number of basis functions
For Lorenz: $\ell = 10$ (1 constant + 3 linear + 6 quadratic)

Coefficient Matrix $\boldsymbol{\Xi}$

Each column ξ_i gives coefficients for the i-th state equation:

$$\boldsymbol{\Xi} = \begin{bmatrix} \xi_{0,1} & \xi_{0,2} & \cdots & \xi_{0,n} \\ \xi_{1,1} & \xi_{1,2} & \cdots & \xi_{1,n} \\ \vdots & \vdots & & \vdots \\ \xi_{\ell,1} & \xi_{\ell,2} & \cdots & \xi_{\ell,n} \end{bmatrix}_{\ell \times n}$$

Interpretation: The i-th dynamical equation is

$$\dot{u}_i = \sum_{j=1}^{\ell} \xi_{j,i} \theta_j(\mathbf{u})$$

Sparsity: Most entries of $\boldsymbol{\Xi}$ should be zero!

Example: If $\dot{x} = -10x + 10y$, then

$$\xi_{\text{for } \dot{x}} = [0, -10, 10, 0, 0, 0, \ldots, 0]^T$$

Overview of SINDy Algorithm

1. Collect data $\mathbf{U}$

2. Compute derivatives $\dot{\mathbf{U}}$

3. Build library $\boldsymbol{\Theta}(\mathbf{U})$

4. Sparse regression: find $\boldsymbol{\Xi}$

5. Extract equations

Step 1-3: Data Preparation

1. **Collect Data**
   - Measure system at times $t_1, t_2, \ldots, t_m$
   - Store in matrix $\mathbf{U}$ (size $m \times n$)
2. **Compute Derivatives**
   - Finite differences: $\dot{u}(t_i) \approx \frac{u(t_{i+1}) - u(t_i)}{\Delta t}$
   - Or use higher-order methods
   - Store in matrix $\dot{\mathbf{U}}$ (size $m \times n$)
3. **Build Library**
   - Choose basis functions: polynomials, trig, etc.
   - Evaluate at each data point
   - Store in matrix $\mathbf{\Theta}(\mathbf{U})$ (size $m \times \ell$)

Step 4: Sparse Regression

Problem: Solve $\dot{\mathbf{U}} \approx \boldsymbol{\Theta}(\mathbf{U})\boldsymbol{\Xi}$ with sparse $\boldsymbol{\Xi}$

This is n separate sparse regression problems (one per column):

$$\dot{\mathbf{u}}_i \approx \boldsymbol{\Theta}(\mathbf{U})\xi_i \quad \text{for } i = 1, \ldots, n$$

Methods:

- **LASSO:** $\min_\xi \|\dot{\mathbf{u}} - \boldsymbol{\Theta}\xi\|_2^2 + \lambda\|\xi\|_1$
- **Ridge:** $\min_\xi \|\dot{\mathbf{u}} - \boldsymbol{\Theta}\xi\|_2^2 + \lambda\|\xi\|_2^2$
- **Elastic Net:** Combination of LASSO + Ridge
- **STLSQ:** Sequential Thresholded Least Squares (SINDy default)

Why not plain least squares?

- Would give dense solution (all coefficients non-zero)
- Overfitting and no interpretability

STLSQ: Sequential Thresholded Least Squares

Algorithm:

1. Solve least squares: $\xi \leftarrow \arg\min_\xi \|\dot{\mathbf{u}} - \boldsymbol{\Theta}\xi\|_2^2$
2. Threshold: Set $\xi_j = 0$ if $|\xi_j| < \lambda$
3. Re-solve using only active terms (non-zero ξ_j)
4. Repeat steps 2-3 until convergence

Why STLSQ?

- Faster than LASSO for large problems
- More stable numerically
- Simple and effective

Key parameter: Threshold λ

- Too small: not sparse enough, overfitting
- Too large: miss important terms, underfitting

STLSQ Example

Consider $\dot{x} = -2x$ with library $[1, x, x^2, x^3]$

Iteration 0: Least squares solution

$$\xi = [0.05, -2.01, -0.03, 0.02]^T$$

Iteration 1: Threshold at $\lambda = 0.1$

$$\xi = [0, -2.01, 0, 0]^T \quad \text{(set small values to 0)}$$

Iteration 2: Re-solve with only x term active

$$\xi = [0, -2.00, 0, 0]^T$$

Iteration 3: Converged! (no change)

Discovered equation: $\dot{x} = -2x$ (exact!)

Example 1: Linear System

True system:

$$\dot{x} = -2x$$
$$\dot{y} = y$$

Setup:

- Initial condition: $(x_0, y_0) = (3, 0.5)$
- Sample 100 time points over $t \in [0, 1]$
- Polynomial library with degree 3
- STLSQ with threshold $\lambda = 0.2$

SINDy discovers:

$$\dot{x} = -2.000x$$
$$\dot{y} = 1.000y$$

Perfect recovery!

Example 2: Lorenz System

True system: $(\sigma = 10, \rho = 28, \beta = 8/3)$

$$\dot{x} = 10(y - x)$$
$$\dot{y} = x(28 - z) - y = 28x - xz - y$$
$$\dot{z} = xy - \frac{8}{3}z$$

SINDy discovers: (threshold $\lambda = 0.025$)

$$\dot{x} = -10.005x + 10.004y$$
$$\dot{y} = 27.805x - 0.958y - 0.993xz$$
$$\dot{z} = -2.667z + 0.999xy$$

Very close! Small errors due to:

- Numerical differentiation noise
- Finite data

Lorenz: Library Comparison

With polynomial library (degree 2):

- Sparse, accurate solution
- Correct terms identified

With Fourier library (sines and cosines):

$$\dot{x} = 0.772\sin(x) + 2.097\cos(x) - 2.298\sin(y) - 3.115\cos(y)$$
$$\dot{y} = 1.362\sin(y) - 0.222\cos(y)$$

- Not sparse!
- Poor approximation
- Many terms with similar magnitudes

Lesson: Library choice matters! Must match system's structure.

PySINDy: Python Implementation

Three main components:

1. `pysindy.differentiation`: Compute $\dot{\mathbf{U}}$ from $\mathbf{U}$
2. `pysindy.feature_library`: Build $\mathbf{\Theta}(\mathbf{U})$
3. `pysindy.optimizers`: Sparse regression for $\mathbf{\Xi}$

Basic workflow:

```
import pysindy as ps

# Define components
model = ps.SINDy(
    differentiation_method=ps.FiniteDifference(),
    feature_library=ps.PolynomialLibrary(degree=2),
    optimizer=ps.STLSQ(threshold=0.2)
)

# Fit to data
model.fit(X, t=t)
```

PySINDy Example: Linear System

```
import numpy as np
import pysindy as ps

# Generate data
t = np.linspace(0, 1, 100)
x = 3 * np.exp(-2 * t)
y = 0.5 * np.exp(t)
X = np.stack((x, y), axis=-1)

# Create and fit model
model = ps.SINDy(
    feature_library=ps.PolynomialLibrary(degree=3),
    optimizer=ps.STLSQ(threshold=0.2),
    feature_names=["x", "y"]
)
model.fit(X, t=t)
```

Simulation and Prediction

Once model is fit, can simulate forward in time:

```
# New initial condition
x0_new = [6, -0.1]
t_test = np.linspace(0, 1, 100)

# Simulate
x_pred = model.simulate(x0_new, t=t_test)

# Plot
plt.plot(x_pred[:, 0], x_pred[:, 1], 'r--')
plt.xlabel('x')
plt.ylabel('y')
```

This integrates the discovered ODEs!

- Uses discovered $\dot{\mathbf{u}} = \mathbf{\Theta}(\mathbf{u})\xi$
- Standard ODE solver (e.g., RK45)
- Can predict outside training window

Handling Noisy Data

Problem: Real data has measurement noise

$$\mathbf{u}_{\text{measured}} = \mathbf{u}_{\text{true}} + \epsilon$$

Challenges:

- Numerical differentiation amplifies noise
- Spurious terms may appear in $\mathbf{\Xi}$
- Need robust differentiation and regression

Solutions:

1. **Better derivatives:**
   - Total variation regularization
   - Polynomial smoothing + differentiation
   - Direct measurement of derivatives (if possible)
2. **Ensemble methods:**
   - Bootstrap: fit on multiple data subsets
   - Keep only consistently identified terms
3. **Regularization:**

Partial Differential Equations (PDEs)

Extension: SINDy can discover PDEs!

$$\frac{\partial u}{\partial t} = \mathcal{N}[u]$$

where $\mathcal{N}$ is a nonlinear differential operator.

Example: Burgers' Equation

$$\frac{\partial u}{\partial t} = -u\frac{\partial u}{\partial x} + \nu\frac{\partial^2 u}{\partial x^2}$$

Approach:

- Spatio-temporal data: $u(x_i, t_j)$
- Library includes spatial derivatives: $u, u_x, u_{xx}, uu_x, \ldots$
- Compute derivatives: u_t, u_x, u_{xx} numerically
- Apply SINDy: $u_t \approx \Theta(u, u_x, u_{xx}, \ldots)\xi$

Result: Discovers PDE from spatio-temporal snapshots!

Weak Formulation (Weak SINDy)

Problem: Computing derivatives is noisy

Solution: Use weak formulation (integral form)

$$\int_0^T \dot{u}(t)\phi(t)dt = \int_0^T f(u(t))\phi(t)dt$$

for test functions $\phi(t)$.

Integration by parts:

$$-\int_0^T u(t)\dot{\phi}(t)dt + [u\phi]_0^T = \int_0^T f(u(t))\phi(t)dt$$

Advantage: No need to compute $\dot{u}$ explicitly!

- Differentiation transferred to test functions
- More robust to noise
- Better for conservation laws

Constrained SINDy

Incorporate prior knowledge:

1. **Conservation Laws**
   - Energy conservation: $\frac{d}{dt}(T + V) = 0$
   - Add constraints to optimization
2. **Symmetries**
   - Rotational invariance
   - Time-reversal symmetry
   - Restrict library to symmetric functions
3. **Physical Constraints**
   - Positivity: some states must be > 0
   - Boundedness: states remain in physical range
   - Add inequality constraints

Implementation:

$$\min \|\dot{\mathbf{u}} - \mathbf{\Theta}\xi\|_2^2 + \lambda\|\xi\|_1 \quad \text{subject to } A\xi = b, C\xi \leq d$$

Neural Network Libraries

Problem: Don't know what basis functions to use

Solution: Learn library with neural networks!

$$\dot{\mathbf{u}} = \boldsymbol{\Xi} \cdot \boldsymbol{\Theta}_{NN}(\mathbf{u}) \tag{3}$$

where $\boldsymbol{\Theta}_{NN}(\mathbf{u})$ is output of a neural network.

Approach:

1. Initialize $\boldsymbol{\Theta}_{NN}$ (e.g., 2-layer MLP)
2. Alternate:
    - Fix $\boldsymbol{\Theta}_{NN}$, optimize $\boldsymbol{\Xi}$ (sparse regression)
    - Fix $\boldsymbol{\Xi}$, optimize $\boldsymbol{\Theta}_{NN}$ (gradient descent)
3. Converge to sparse, expressive representation

Advantages:

- Discovers appropriate basis functions
- More flexible than fixed library
- Still maintains interpretability through sparsity

Applications

1. **Fluid Dynamics**
   - Turbulence modeling
   - Reduced-order models for CFD
2. **Biology**
   - Gene regulatory networks
   - Population dynamics
   - Neuroscience (neural dynamics)
3. **Chemical Kinetics**
   - Reaction mechanisms
   - Combustion modeling
4. **Climate Science**
   - Atmosphere-ocean dynamics
   - Reduced climate models
5. **Engineering**

Limitations

1. **Library Choice**
   - Success depends on choosing appropriate basis functions
   - May miss complex nonlinearities not in library
2. **Data Requirements**
   - Needs sufficient data coverage of state space
   - Poorly sampled regions lead to poor identification
3. **Noise Sensitivity**
   - Numerical differentiation amplifies noise
   - May require careful preprocessing
4. **Parameter Tuning**
   - Threshold λ is problem-dependent
   - No universal rule for selection
   - Requires validation and experimentation
5. **Interpretability vs. Accuracy**

Comparison: SINDy vs. Neural ODEs

	SINDy	**Neural ODE**
Form	$\dot{u} = \Theta(u)\xi$ (sparse)	$\dot{u} = NN(u, \theta)$ (black box)
Interpretability	High (symbolic)	Low (weights)
Data	Needs derivatives	Just states
Training	Fast (regression)	Slow (backprop + ODE solve)
Extrapolation	Good if library matches	Can be poor
Flexibility	Limited by library	Very flexible
Use case	Discovery	Prediction

Complementary approaches!

- Use SINDy when interpretability matters

Best Practices

1. **Data Collection**
   - Ensure good coverage of state space
   - Sample densely enough for accurate derivatives
   - Multiple trajectories better than one long trajectory
2. **Preprocessing**
   - Normalize/scale variables appropriately
   - Denoise if possible before differentiation
3. **Library Selection**
   - Start with polynomials (universal approximators)
   - Add domain-specific functions if known
   - Use cross-validation to compare libraries
4. **Validation**
   - Test on held-out data
   - Simulate discovered equations and compare
   - Check physical plausibility

Summary

SINDy: Sparse Identification of Nonlinear Dynamics

Key Ideas:

1. Physical laws are typically **sparse**
2. Represent dynamics as: $\dot{\mathbf{u}} = \boldsymbol{\Theta}(\mathbf{u})\boldsymbol{\Xi}$
3. Use **sparse regression** (STLSQ) to find $\boldsymbol{\Xi}$
4. Discovers interpretable, symbolic equations from data

Advantages:

- Interpretable results (symbolic equations)
- Fast and efficient
- Works with modest data
- Generalizes well (if library is appropriate)

Limitations:

- Needs good library choice
- Sensitive to noise in derivatives
- Requires parameter tuning

References

Brunton, S. L., Proctor, J. L., & Kutz, J. N. (2016).
Discovering governing equations from data by sparse identification of nonlinear dynamical systems.
Proceedings of the National Academy of Sciences, 113(15), 3932-3937.

Champion, K., Lusch, B., Kutz, J. N., & Brunton, S. L. (2019).
Data-driven discovery of coordinates and governing equations.
Proceedings of the National Academy of Sciences, 116(45), 22445-22451.

Rudy, S. H., Brunton, S. L., Proctor, J. L., & Kutz, J. N. (2017).
Data-driven discovery of partial differential equations.
Science Advances, 3(4), e1602614.

de Silva, B., Champion, K., Quade, M., Loiseau, J. C., Kutz, J. N., & Brunton, S. L. (2020).
PySINDy: A Python package for sparse identification of nonlinear dynamical systems from data.

Questions?

Thank you!

SINDy: Discovering the equations of nature from data

Resources:

- PySINDy: `https://github.com/dynamicslab/pysindy`
- Original paper: Brunton et al., PNAS 2016
- Tutorial notebooks in course repository